Chimera Workshop

September 4th, 2002

GriPhyN
Data Intensive Science

Outline

- Install Chimera
- Run Chimera
  - Hello world
  - Convert simple shell pipeline
  - Some diamond, etc.
- Get (some) user apps to run with *shplan*
- Introduce The Grid
  - As time permits

Prepare Chimera Installation

- Log onto your Linux system.
- Ensure Java version 1.3.*
  - Type "$JAVA_HOME/bin/java –version"
  - If necessary, download from http://java.sun.com/j2se/1.3/
  - Use Java 1.4.* at your own risk
- Set JAVA_HOME environment variable
  - Points to Java installation directory
  - Example: setenv JAVA_HOME /usr/lib/java

Install Chimera

- Download the latest tar-ball
  http://www.griphyn.org/workspace/VDS/snapshots.php
  - The binary release is smaller
  - The source release allows patching
  - Let's try the source release for now…
- Unpack
  - *gtar xvzf vds-source-YYYYMMDD.VV.tar.gz*
- *cd vds-1.0b1*
  - Current version is beta1

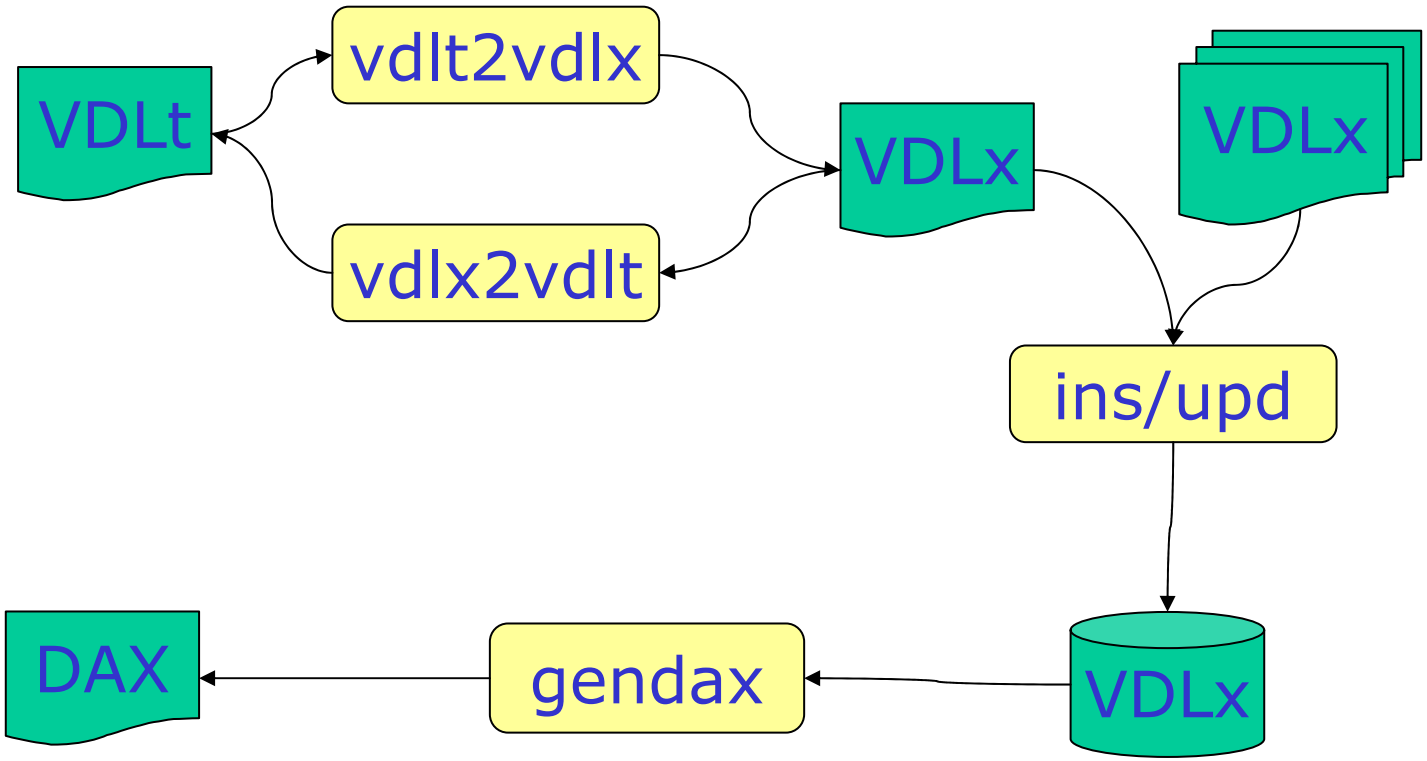
Starting Chimera

- Source necessary shell script (all releases)
  - Some environments require explicit setting of *VDS_HOME=`/bin/pwd`* before sourcing
  - Bourne: *. set-user-env.sh*
  - C-Shell: *source set-user-env.csh*
  - Ignore warnings (not errors) about VDS_HOME
- Run *test-version* program to check
  - Script is a small sanity test
  - If it fails, Chimera will not run!

Sample *test-version* Script Output

```
# checking for JAVA_HOME
# checking for CLASSPATH
# checking for VDS_HOME
# script JAVA_HOME=/usr/lib/java
# script CLASSPATH=…/lib/java-getopt-
1.0.9.jar:…/lib/xercesImpl.jar:…/lib/chimera.jar:…/lib/antlrall
.jar:…/lib/xmlParserAPIs.jar
# starting java
# Using recommended version of Java, OK
# looking for Xerces
# found xercesImpl\.jar, OK for now
# looking for antlr
# found antlr(all)?.jar, OK for now
# looking for GNU getopt
# found java-getopt-.\..\..\.jar, OK for now
# looking for myself
# in developer mode, ignore not finding chimera.jar
# found chimera\.jar, OK for now
```

Working with Chimera files



Only the abstract planning process is shown

The Virtual Data Languages

- VDLt is textual
- Concise
- For human consumption
- Usually for (TR) **transformation**
- Process by converting to VDLx

- VDLx uses XML
- Uses XML-Schema
- For generation from scripts
- Usually for (DV) **derivations**
- Storage representation of current VDDB

Virtual Data Language

- A "function call" paradigm
  - Transformations are like function definitions.
  - Derivations are like function calls.
- Many calls to one definition
  - Many (zero to N) calls of the same transformation.

Hello World

- Exercise 1:
  - Wrap *echo 'Hello world!'* into VDL
- Start with the transformation

```
TR hw( output file )
   {argument = "Hello world!";
    argument stdout = ${file}; }
```

- Then "call" the transformation

```
DV d1->hw( file=@{output:"out.txt"} );
```

- Save into a file *hw.vdl*

What to do with it?

- All VDLt must be converted into VDLx before it becomes "usable" to the VDS.

  vdlt2vdlx hw.vdl hw.xml

- All VDLx must be stored into your VDDB, before the system can work with it

  insertvdc –d my.db hw.xml

- or

  updatevdc –d my.db hw.xml

From VDC to DAX

- Deriving the provenance of a logical file or derivation is the *abstract planning* process.
- All files and transformations are logical!
  - The complete provenance will be unrolled
  - No external catalogs (TC,RC) are queried
- Ask the catalog for the produced file

  gendax –d my.db –l hw –o hw1.dax –f out.txt
- Or ask for the derivation (to be fixed)

  gendax –d my.db –l hw –o hw2.dax –i hello

The DAX file

- The result of the Chimera abstract planner.
- Richer than Condor DAGMan format.
- Expressed in terms of logical entities.
- Contains complete (build) lineage.
- Consists of three major parts
  1. All logical files necessary for the DAG.
  2. All jobs necessary to produce all files.
  3. The dependencies between the jobs.

The Transformation Catalog

- Translates logical transformation into an application specific for a certain pool
- Similar in all versions of Chimera
- Simple, text based file
  - 3+ columns
  - Blank lines and comments are ignored
- Standard location
  - $VDS_HOME/var/tc.data
  - adjustable with property vds.db.tc

Transformation Catalog Columns

- 1st column is the pool handle
  - Shell planner only uses the "local" handle
  - Other handles for concrete planner
- 2nd column is the logical transformation
  - Format: <ns>_ _ <name>_<version>
  - Name-only names are just <name>
- 3rd column is the path to the executable
- 4th column for environment variables
  - Use "null" if unused
  - Format: key=value;key=value

The Shplan Replica Catalog

- A replica catalog translates a logical filename into a set of physical filenames
- This RC is for the shell planner only
- Simple textual file
  - Three columns
  - Blank lines and comments are ignored
- Standard location
  - $VDS_HOME/var/rc.data
  - Adjustable with property vds.db.rc

Shell planner RC Columns

- 1st column is the pool handle
  - Shell planner only uses the "local" handle
  - Other handles for concrete planner
- 2nd column is the logical filename
  - A LFN may contain slash etc.
- 3rd column is the path to the file
- Are multiples allowed?
  - No, only the last (first?) match is taken

TC and RC Short-cuts

- Application locations can come from VDL
  - hints.pfnHint takes precedence over TC
  - Shell planner may run w/o any TC
- Files need not be registered with RC
  - Existence checks are done in file system
  - RC updates can be optional
  - Shell planner may run w/o any RC

➨ After all, we run locally with the shell planner!

Preparing to Run The Shell Planner

- Make sure that your TC contains a translation for logical transformation "hw"

  local hw /bin/echo null

- Make sure that there is a RC without weird content:

  cp /dev/null $VDS_HOME/var/rc.data

Running The Shell Planner

- Run the shell planner

  shplanner –o hw hw1.dax

- Check directory "hw"
  - Master script: <DAX-label>.sh
  - Job scripts: <TR>_<DAX-JobID>.sh
  - Helper files: <TR>_<DAX-JobID>.lst

Running The Shell-Plan

- Shell planner generates a master plan
  - This is a tool-kit ➠ no auto-run feature
- Run the master plan

  (cd hw && ./hw.sh)
- Can check log file for status etc.
  - Standard name: <DAX-label>.log
- Master script will exit with 0 on success
  - Exit 1, if any sub-script fails.

Convert A Unix Pipeline

- Example 2: Full name from a username

  grep ^user /etc/passwd | awk –F: '{ print $5 }'

1. Create abstract transformations
2. Create concrete derivations to call them
3. Convert into VDLx
4. Add to VDDB
5. Run abstract planner
6. Run shell planner
7. Run master script

Abstract Transformations

- Grep for an arbitrary user name

```
TR grep( none name, output of )
  { argument = "^"${name}" /etc/passwd";
  argument stdout = ${of}; }
```

- Extract a full name from a line

```
TR awk( input line, output full )
  { argument stdin = ${line};
  argument stdout = ${full};
  argument = "-F: '{ print $5 }' "; }
```

Concrete Derivations

- Run with a concrete username…

```
DV d2->grep( name="voeckler",
  of=@{output:"grepout.txt"} );
```

- … and post-process the results

```
DV d3->awk( line=@{in:"grepout.txt"},
  full=@{out:"awkout.txt"} );
```

- The two derivations are linked by a LFN
  - "grepout.txt" is produced by "d2"
  - "grepout.txt" is consumed by "d3"

Convert, Insert and Plan

- Convert into VDLx

  vdlt2vdlx ex2.vdl ex2.xml

- Insert into your VDDB

  insertvdc –d my.db ex2.xml

- Run abstract planner

  gendax –d my.db –l ex2 –o ex2.dax –f awkout.txt

Run Shell Planner

- Prepare transformation catalog

  local grep /usr/bin/egrep null

  local awk /opt/gnu/bin/gawk null

- Run shell planner

  shplanner –o ex2 ex2.dax

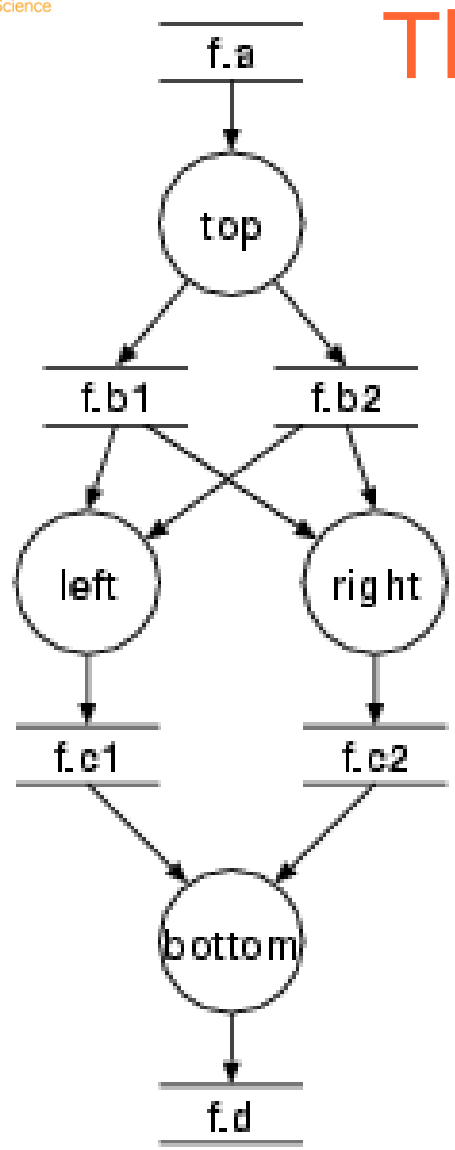
- Run master plan

  (cd ex2 && ./ex2.sh)

Kanonical Executable for Grids

- Simple application
  - Copies all input with indentation to all output files
  - Adds additional data about itself
- Allows tracking
  - What ran where, when, which architecture, and how long
- To be used as stand-in for real applications
  - Allows to check the DAG stages

The Diamond DAG



- Complex structure
  - Fan-in
  - Fan-out
  - "left" and "right" can run in parallel
- Uses input file
  - Register with RC
- Complex file dependencies

The Diamond DAG II

- The "black diamond" takes one input file

  echo –e "local\tf.a\t${HOME}/f.a" »
  $VDS_HOME/var/rc.data

  date > ${HOME}/f.a

- It uses three different transformations

  echo –e "local\tpreprocess\t$VDS_HOME/bin/keg\tnull"
  » $VDS_HOME/var/tc.data

  echo –e "local\tfindrange\t$VDS_HOME/bin/keg\tnull"
  » $VDS_HOME/var/tc.data

  echo –e "local\tanalyze\t$VDS_HOME/bin/keg\tnull" »
  $VDS_HOME/var/tc.data

A Big Example

- Overall tree structure
  - + 70 "regular" diamond DAGs on top
  - + 10 collecting nodes
  - + 1 final collector
  - = 291 jobs to run
- All files are generated, no RC preloading
- Add "multi" TR to last example

echo –e "local\tmulti\t$VDS_HOME/bin/keg\tnull" »
$VDS_HOME/var/tc.data